Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

CRT-2003-22

A NETWORK MANAGEMENT FRAMEWORK USING MOBILE AGENTS

by

Jonathan Lefebvre
Steven Chamberland
Samuel Pierre

September 2003

Université de Montréal
C.P. 6128, succursale Centre-ville, Montréal QC H3C 3J7 Canada
Téléphone: (514) 343-7575 / Télécopieur: (514) 343-7121

crt@crt.umontreal.ca
www.crt.umontreal.ca

HEC ÉCOLE POLYTECHNIQUE MONTRÉAL Université de Montréal

104-00251/

PUBLICATION
CRT-2003-22

A NETWORK MANAGEMENT FRAMEWORK USING MOBILE AGENTS

by

Jonathan Lefebvre [1]
Steven Chamberland [1,2]
Samuel Pierre [1]

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1. Computer Engineering Department, École Polytechnique de Montréal, C.P. 6079, succursale Centre-ville, Montréal, Canada H3C 3A7 (jonathan-2.lefebvre, steven.chamberland, samuel.pierre@polymtl.ca)

2. Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec
Bibliothèque nationale du Canada, 2003

Abstract

In this paper, we present a framework that has the ability to perform network management tasks on heterogeneous networks using mobile agents. Our framework handles the inability of many network devices to run mobile agents. To cope with fast evolving networks, we present a modular approach that provides tools to build management mobile agents quickly to perform virtually any task. While the main focus of the project is the framework, we present an example of mobile agents that are able to locate a fixed set of network failures and detect the possible causes accurately.

Keywords: network management; intelligent agents; fault tolerance.

Résumé

Dans cet article, nous présentons un modèle qui permet l'exécution de tâches de gestion dans des réseaux hétérogènes en utilisant des agents mobiles intelligents. Notre modèle fonctionne dans le contexte où des éléments de réseau ne permettent pas nécessairement l'exécution d'agents mobiles. Pour contrer l'effet de l'évolution rapide des réseaux, nous proposons une approche modulaire qui permet le développement rapide d'agents mobiles de gestion. Finalement, nous présentons un exemple d'application à la gestion des fautes dans les réseaux.

Mots-clés: gestion des réseaux; agents intelligents; tolérance aux pannes.

1. Introduction

Mobile agents still do not seem to dominate the world of networks with real, convincing and useful applications. Meanwhile, network management of heterogeneous components in large networks is still hard to achieve automatically and efficiently. It is now a fact that networks will not grow using simple and similar technologies but tend to use complex protocols and many different devices from many different vendors.

Mobile agents have the ability to migrate on nodes and cope with different technologies. Also, they decentralize management and distribute load over the network. They also provide a quick and flexible way to create solutions for fast-evolving environments and enable the possibility to automate network management tasks efficiently. While our aim is large-scale networks where many tasks are still executed by humans, these abilities are crucial.

Many research groups have worked on network management solutions using mobile agents. The objective of the JAMES project [20] is to create an efficient mobile agent platform mainly used for network management. It provides many optimizations in comparison to commercial mobile agent platforms that are mandatory to achieve efficiency and high performance in the domain of network management. The Network Management and Artificial Intelligence Laboratory of Carleton University [5] is doing research on many applications and models to bring mobile agents to network management. Their aims are large and they have studied many aspects of the subject such as the need for a uniform management interface, the use of the simple network management protocol (SNMP) [13], fault location [22] and intelligent agents [22]. Novel architectures inspired by simple life organisms have been proposed. One such architecture is called ECOMOBILE [16]. It uses mobile agents to execute task objectives, but these agents are not by themselves network tasks. They have a life, compete with each other, die,

exchange, take or leave task objectives at any time. This architecture offers an interesting way to regulate its mobile agent population while achieving network management tasks. ANTNET [7] is an architecture in that field. It was introduced at first to use mobile agents for adaptive routing. However, it has inspired a lot of research [2,19,21]. The common point of this research is the accomplishment of complex objectives using simple mobile agents. On the concept of proximity, which we will introduce later, a research group [12] has studied efficient ways to place mobile agents on a network combining both mobility and remote monitoring. Monitoring is one part of network management and may be used for fault management as well.

In Section 2, we present the aims of the project and the framework. After that, in Section 3, we will present implementation details and mobile agents used to achieve real network management tasks in a heterogeneous network. In Section 4, we present the test networks and results acquired with the diagnostic agent we created. Finally, we conclude with a brief summary, limitations and future work.

2. The Framework

The framework encompasses the tools and functions used by mobile agents and their environment. It is built to run on top of existing network management systems as a solution to enhance current installations capabilities. In this section, we will discuss the main goals of the framework as well as its key parts.

2.1. Goal and aims

As stated in the introduction, our goal was to create a network management framework using mobile agents. Our primary interest was to prove the utility of them in real networks and applications. This is, from our point of view, a first step in validating efforts done on mobile agents and network management. Because of this, the framework must be able to cope with

network components that cannot execute and store mobile agents. Also, it was important to show the ability of these agents to manage heterogeneous devices. One of our goals was to give agents as little information on their environment as possible, enabling them to manage an evolving network while keeping a small size.

Another important aspect is performance. Mobile agents should not impose a much higher overhead on the network than their client-server counterparts. Performance is then an important goal and is achieved in part by not letting mobile agents carry the network topology or nodes information.

2.2. Model

Our framework can easily integrate multiple mobile agents in existing networks by using an expert system and using existing management facilities.

The model suggests using typically one or two agents per network management task. This allows us to limit inter-agent communications that can be costly if the framework tends to use small and inefficient agents. By inefficient, we mean agents that cannot act autonomously. In technical reports by [17] and [18], it was shown that using only one mobile agent for one task increases the task response time, but lowers the total traffic on the network. A paper presenting a real configuration task [4] provides a performance evaluation of using one mobile agent against using parallel mobile agents for one task. In fact, communication and synchronization between multiple agents slows the whole task to a point where a single agent performs better in both areas. It is therefore impossible to give an optimal choice for every topology, network size and management task. Our choice to use few mobile agents for one task is based on [4], on our belief that most management tasks are complex, and on our motivation to limit the network load

and agent building complexity. For tasks where there are few dependencies between each device, we suggest using multiple instances of the same mobile agent to divide the task.

Network management is done using network tools already available. The advantage of using existing tools such as SNMP is pointed out by [24]. They are already widely supported and implemented in network devices and limit the effort involved in building a network management framework. Also, mobile agents tend to use these tools more efficiently and in new ways. We present the important aspects of the framework in this section.

2.2.1. Concepts

A place where network management is done is called a management station. In mobile agent terms, it could also be named an agency, a place or a region, and must be able to run a mobile agent platform. This usually implies a station that could run a Java virtual machine. A component or device is what the mobile agent has to manage. It may also be a management station. A network station could manage itself, being itself a component. The framework was built with these concepts in mind.

Mobile agents are entities that have the ability to migrate with their code while preserving their data and execution state. In this framework, mobile agents are named management mobile agents because they have network management capabilities.

2.2.2. Management Table

The management table is the only knowledge of the network that the framework provides. Any other knowledge is taken from the network as needed by mobile agents. This table keeps links between network components and network management stations. One important utility of this table is to manage components that cannot receive mobile agents. For our experiments, associations in this table were static, meaning that one management station was bind

permanently to one or many components. The association is based on proximity. Proximity may be determined by a wide selection of factors. In our case, it is simply the number of hops between the station and the component. It could have been a cost function based on any measurement or factor to optimize further associations and station placements. Static associations do not mean that a component is always managed by the same station. It only tells the mobile agent the preferred management station for a particular component. The framework could use a dynamic update for this table and a way to adapt to network modifications made on topology or behavior. One research study [12] offers interesting ideas on how this aspect could be improved.

For optimization purposes, these tables are installed on each management station, freeing the mobile agent's memory to save bandwidth. This also allows local optimizations when it is not clear whether a central component must be managed by one station or another depending on the point of view.

2.2.3. Network Management Interface

To limit the size of mobile agents and mostly their building complexity, a minimum of uniform management is needed. To allow more flexibility, the framework offers many standard interfaces. Uniform interfaces are key parts of many mobile agent systems such as the MAS (Mobile Agent Server) [9,10], the VMC (Virtual Managed Component) [23] or in the Phoenix architecture [15]. Our framework allows more flexibility and mobile agents are not tied to these interfaces. This lets us introduce two kinds of mobile agents, general agents and specialized agents. The general agent will mostly use uniform interfaces, managing the network with a limited set of tools. The specialized agent is able to do a lot more tasks and use specialized

features, to the expense of complexity and size. What are hidden behind these interfaces are stationary agents, which is the next subsection.

2.2.4. Technology Stationary Agent

Stationary agents are used to implement network management code that has to be dynamic, but may be totally inefficient to move with mobile agents. By dynamic, we mean that they could keep a state, be modified easily, keep local information in cache for fast and efficient retrieving. This code is moved once, and stays permanently on the station.

Technology stationary agents implement a set of uniform management interfaces, which were introduced in the previous subsection. The management table that keeps references between management stations and components also keeps a set of network management abilities for a component. Such abilities could be an operating system application programming interface (API), a protocol like SNMP or any other way to manage a component. These stationary agents look like interface agents found in [8], but fill a wider range of functionalities and utilities. These functionalities may be management tools, native code or any network protocol. An example of an architecture using native libraries is Phoenix [15].

2.2.5. Intelligence

Mobile agents need a great load of intelligence to be able to manage networks of heterogeneous devices. Although the framework uses uniform interfaces, it is still difficult to give agents sufficient intelligence to let them manage these networks confidently. Some research tends to use artificial intelligence [6,11] or collective intelligence [2,7,19,21]. Our focus was to use an expert system, but the framework is not limited to a specific form of intelligence. The network tasks implemented using our framework aim to use proven procedural instructions that are best implemented by an expert system.

2.2.6. Security and Fault-tolerance

The framework is meant for closed networks, meaning that the security level may be lowered. For networks where agents should move on user stations, the framework suggests, but does not yet implement, letting only the approved mobile agent's code to get back from user stations. For confidentiality purposes, mobile agents should give up sensible information from the network before entering a user station. Quotas may also be used to counter flooding.

Since the framework is on top of any network management system, it does not interfere with these systems and is not needed for management. Fault management mobile agents described in next section are able to tell if the network management system is faulty, but they cannot recover completely from such a failure.

2.2.7. Global View of the Framework

We presented the main parts of the framework. In this sub-section, we are interested in providing a global view of the framework. A global view is presented in Figure 1.

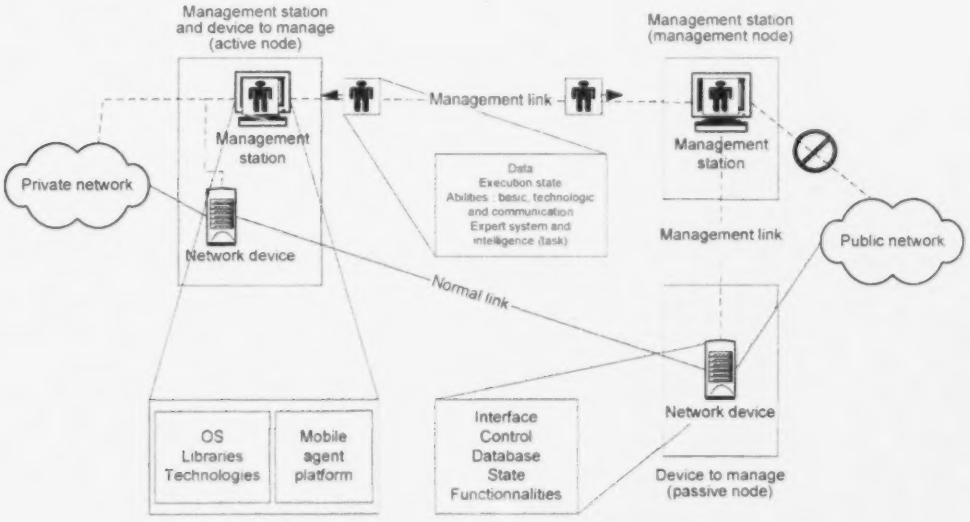

Figure 1. Global view of the framework

Many elements of the framework are detailed in this figure. We have two logical networks: the management network and the normal network. The management network is essentially the management stations and the links between them. The normal network is the part assigned to useful applications. Both logical networks may be the same, have some devices and links shared or be completely different networks. In Figure 1, we see a general view with a network device that can accept a mobile agent (active node) and a device that cannot (passive node). Each passive node must have an associated management station (management node) to be managed. Mobile agents on the network are depicted as a people icon. We also see the composition of a management mobile agent which is mainly its data, its execution state, its intelligence and its abilities. Each management station runs a mobile agent platform and installs basic components and stationary agents used by mobile agents. Mobile agents can migrate in selected private networks and are not allowed to migrate on public networks unless it is in a strictly controlled manner. This requirement serves the minimal security model explained in the previous sub-section. Management stations are detailed in Figure 2.

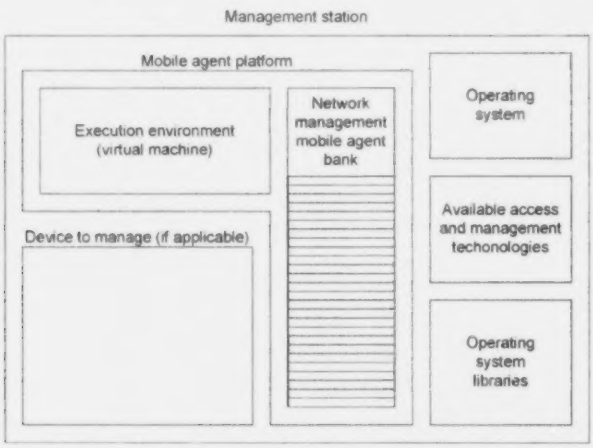

Figure 2. Detailed view of a management station

Each management station of the framework runs a mobile agent platform that can receive and launch mobile agents. A station contains a network management mobile agent bank that stores each mobile agent that may be needed to accomplish a task. These mobile agents, through stationary agents, may access local operating system functions and any ability installed on this station. To go deeper in this model view of a management station, we detailed the mobile agent platform in Figure 3.

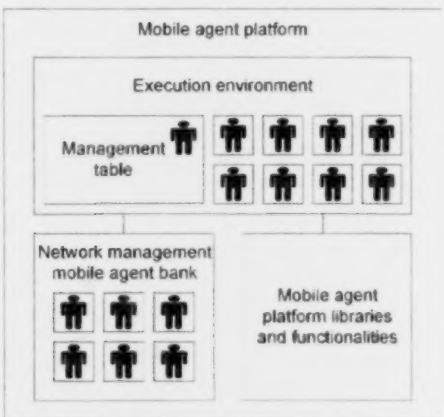

Figure 3. Mobile agent platform

Figure 3 shows that mobile agents can access the management table and can use the functions provided by the mobile agent platform. Access mechanisms and technical details are explained in the implementation section, which is the next section.

3. Implementation

In this section, we first describe all technologies used in the framework and in mobile agents. We have already presented key elements of the framework in the previous section. Implementation of management tables, uniform interfaces, stationary agents and communication between agents are explained in this section. Then, we present in detail mobile agents that were used to validate the framework. We have implemented two mobile agents that are able to find a

set of network failures in a network. These mobile agents never claim to be able to find all possible errors. Since it is an expert system, they should be instructed to do so. The first mobile agent, called *Diagnostic*, tries to go as far as possible in a network to find a failure cause. The diagnostic capability of this agent could be reproduced by a stationary agent. The second agent, named *Search*, is used to pinpoint more precisely the cause of a network failure and exploit mobile agent advantages.

3.1. Technology Used

To implement the framework, we used both Java and C++ programming languages. Java is used for almost all aspects of the framework. The C++ language is used to implement some advanced functions that are accessed using JNI (Java Native Interface). It also demonstrates the framework's ability to use multiple technologies and therefore use virtually any management functionalities in a heterogeneous network. Each mobile agent is implemented only in Java and interacts with Java modules. Mobile agents are built using the Grasshopper platform and API. Grasshopper was a natural choice because of its simplicity, maturity and supported operating systems and implemented standards. To access components using SNMP, we used AdventNet classes and Java Beans.

3.2. Implementation of the Management Tables

Management tables are placed on each network management station. They provide a way for mobile agents implementing network management tasks to plan their next migration. We saw earlier that network management code was, in part, set apart from mobile agents and confined to stationary agents. Management tables are also stationary agents. Because of this nature, they free mobile agents from carrying network topology information and allow local optimizations. A

possible optimization may be obvious in the case depicted in Figure 4. The central switch, depending on the current location of the mobile agent, may be best managed by station A or station B. That case shows one benefit of having tables installed on each station.

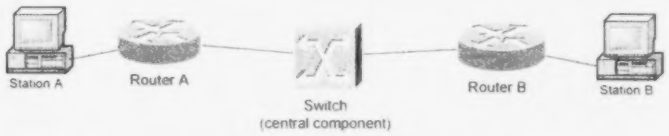

Figure 4. Symmetric topology

The current implementation does not require that a management table be installed on each station, but strongly suggests it. Mobile agents find and access these tables by creating a proxy to the table. We give more information on this type of communication in the next sub-section. The management table implementation uses a hash map to link the management station to network devices. The key of the map is the device unique identifier, therefore allowing a management station to manage any devices and restrict a device to having only one management station assigned per table.

3.3. Communication, Interfaces and Stationary Agents

The main communication medium between agents in our framework is done using remote procedure calls. We use the *GrassHopper* proxy communication mechanism to enable communication between agents. The framework favors local communications between mobile agents and stationary agents on the same place to use as few network resources as possible.

Mobile agents that have to manage the network with a global view use proxy communications to access technologies and functionalities that are hidden inside the stationary agents. They therefore lower their need to carry technology dependant code. These stationary agents are bigger in size because they have to handle each specific technology and component type. The framework favors an installation of these agents on each relevant management station. A mobile

agent knows which functions it can access because it carries the interfaces used to manage the network. To explain how proxy communications are implemented for the framework, we should take a look at Figure 5.

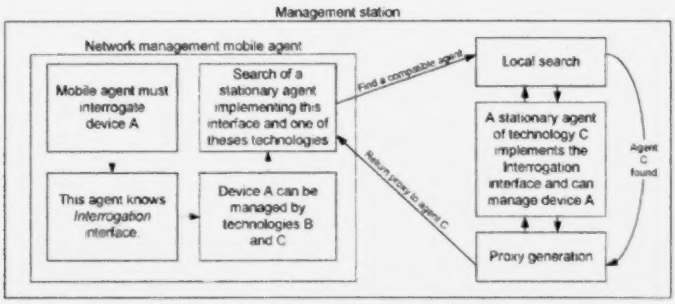
Management station
Network management mobile agent
Mobile agent must interrogate device A
This agent knows Interrogation interface
Search of a stationary agent implementing this interface and one of theses technologies
Device A can be managed by technologies B and C
Find a compatible agent
Return proxy to agent C
Local search
A stationary agent of technology C implements the Interrogation interface and can manage device A
Proxy generation
Agent C found

Figure 5. Lookup mechanism to find a technologic stationary agent

Access to a technologic stationary agent works like a lookup mechanism. For example, let's say that a mobile agent has to manage a device named A. It also knows the interfaces needed for this network management task. The mobile agent then tries to find a stationary agent that implements the interface and has the ability to manage the device A. To do so, it uses a basic set of tests on each stationary agent installed locally. One test, implemented in an interface shared by every agent, allows the mobile agent to test if a stationary agent is able to manage the device. A management station should at least provide one technologic implementation of each interface for each device it has to manage. Otherwise, some devices may not be manageable. Providing more than one technologic implementation can help to counter possible management system errors by giving alternate ways to execute the management task. Interfaces and functions offered by the framework are given in Table 1. This table is not exhaustive, but is a good snapshot of the abilities of the framework.

Table 1. Interfaces and functions of the framework

Parent ability for all abilities	
Functions	**Description**
ManagementStationAvailable	Tell if a component management system is active and may be managed using a given technology

Ability : Base	
Functions	**Description**
Ping	Ping a network address
VerifyService	Verify that a given service is available

Ability : Performance	
Functions	**Description**
GetCongestion	Returns the congestion rate
GetUtilization	Returns the utilization rate

Ability : Routing	
Functions	**Description**
NextComponents	Return the next components physically connected to the device
NextComponent	Return the next component physically connected to the device to join a given destination

Ability : Interrogation	
Functions	**Description**
GetInterfaceAddresses	Returns all addresses of an interface
GetInterfaceInformation	Returns useful information about the interface and its state
GetInterfaceIndex	Returns the interface number given a network address
GetValue	Returns the value of a variable or a state
IsDeviceAnswering	Try to reach the device and return true if it's a success

We used three stationary agents in our framework to manage the test network that will be described in section 5. These agents are listed in Table 2.

Table 2. Stationary agents of the framework and implemented interfaces

Management stationary agents	Implemented interfaces
SSnmp	Interrogation Performance Routing
SBase	Base
SWindows	Interrogation Performance Routing

A stationary agent *SBase* implements server-side functions that are not dependant on the device management technologies. The two other stationary agents implement all the other interfaces.

SSnmp is used for network devices and *SWindows* is used to manage Windows workstations that run mobile agent platforms and are considered as a part of the network to manage.

3.4. Diagnostic Agent

One fact is that a network failure could cause many alarms and cause many direct or indirect failures. The diagnostic mobile agent is used when a connection failure occurs between a source and a destination. It is informed of these two parameters, as well as the port for the connection and nothing more.

The diagnostic agent never stops on the first failure. For this reason, its first task is not to diagnose, but accumulate a series of proofs containing facts and places where these proofs are found. Then, at the end of the *proof finding* phase, it can establish a diagnostic. The *proof finding* phase ends when the diagnostic agent is unable to move further, has moved on or near the destination or has no clues on how to continue (management system down or no route to host). Before terminating, it may try to launch a search agent that returns with an alternate path to the next component. If this agent takes too long, a timeout tells the diagnostic agent to continue without waiting longer. The last phase is called the *diagnostic* phase. Theses phases are more detailed in Figure 6. On process charts, a white box inside a gray box indicates the start of the algorithm while a white box inside a white one indicates a possible end. Grayed boxes indicate algorithm portions that are detailed later in figures 7 and 8.

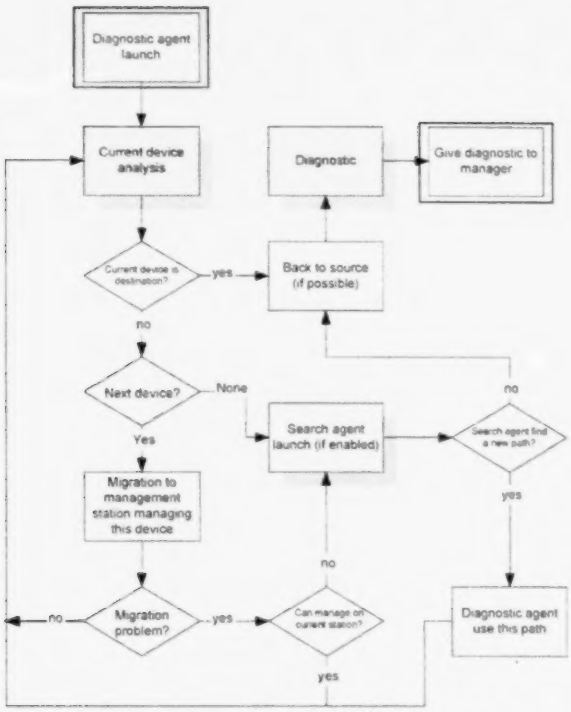

Figure 6. Diagnostic agent global algorithm

The proof finding phase starts when the algorithm starts, it ends when the *Diagnostic* phase is reached and it may be suspended when the mobile agent uses an alternate path given by the search agent. The first thing that the diagnostic agent does is a full analysis of the current device. It then tries to know which device is next on the path between source and destination. By asking the management table, the mobile agent can know which management station is responsible for this device and it tries to migrate on that station. If it is a success, the mobile agent restarts its analysis on the current device and the new management station. If it's not, it tries to manage the device from its current management station. If successful, the algorithm restarts to perform analysis. If not, this tells the diagnostic agent that it has reached a point where it cannot obtain more information. It then has the option of establishing a diagnostic or launch a mobile agent to help find an alternate route (search agent explained in next sub-section). The analysis step is

described in Figure 7. This phase is dependant on which network failures we want to be able to find. The mobile agent does a series of tests without trying to diagnose. A possible optimization here would be to not let the mobile agents run superfluous tests. For now, this phase has no intelligence.

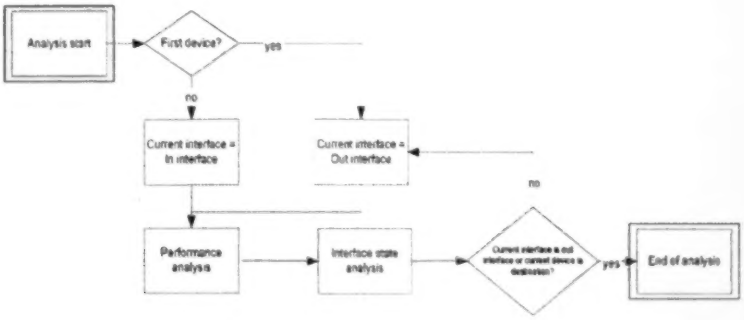

Figure 7. Detailed analysis phase

The analysis phase examines a series of facts. These facts are collected in a *proof list* which is inserted in a *path list*. The *path list* is built by collecting information on each interface on the real path between the source and destination. The next detailed phase is the diagnostic phase presented in Figure 8.

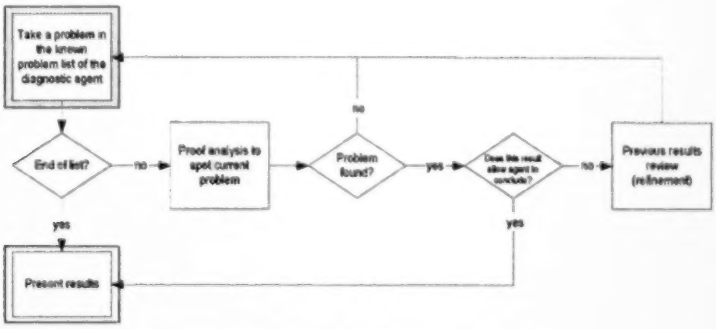

Figure 8. Detailed diagnostic phase

The mobile agent intelligence is mostly concentrated in this phase. It tries, using refinement and testing known cause with collected proofs, to know the best location and possible cause that fit current facts about the network. This diagnostic is usually more precise if more facts are found

about the problem. It never assumes that the last fact collected is the more relevant for the location or the problem. Figure 9 shows how information is collected.

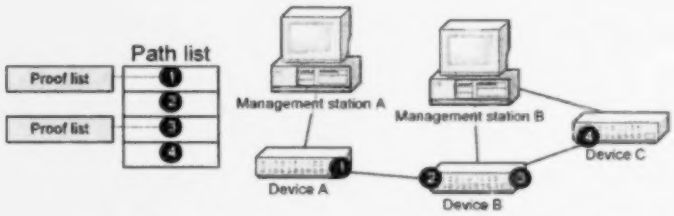

Figure 9. Mobile agent view of the network

Each interface on the path between device A and C is included in the path list. Each time the diagnostic mobile agent finds a possible problem, it includes a proof list for that interface. In our example, facts are found for interfaces 1 and 3.

3.5. Search Agent

The search agent clones itself on each route it finds on a given node. Its goal is to find the destination using another path in the network that routing tables may not contain. When it finds the destination, it then tries to come back to the source using routing tables. When it finds a point where it cannot move using these tables, this lets it know that this may be the other end that the diagnostic agent was trying to reach. It then reuses the alternate path to come back to the place where the diagnostic agent is, to give it the extra information. The diagnostic agent then suspends its *proof finding* phase to move to the component found by the search agent using the alternate path. Arrived at destination, it restarts its *proof finding* phase. This is a summary of the complete algorithm found in Figure 10.

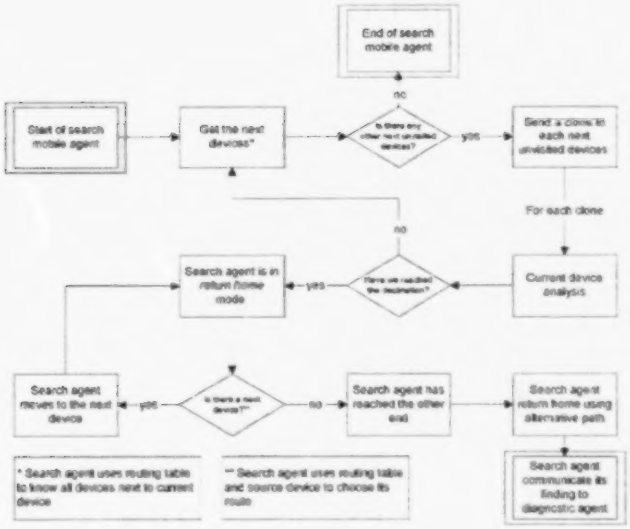
End of search mobile agent
no
Start of search mobile agent
Get the next devices*
yes
Send a clone to each next unvisited devices
For each clone
no
Search agent is in return home mode
yes
Current device analysis
Search agent moves to the next device
yes
no
Search agent has reached the other end
Search agent return home using alternative path
Search agent communicate its finding to diagnostic agent
* Search agent uses routing table to know all devices next to current device
** Search agent uses routing table and source device to choose its route

Figure 10. Search agent algorithm

In case the search agent never returns, the diagnostic agent is still able to give a good estimation of the problem just like a remote management solution. The *Search* agent is an addition that takes advantage of multi-path networks. However, it is costly in traffic. To limit its spawn, a hop counter is implemented to terminate itself after too many jumps. This maximum hop value should be set carefully according to network scale and desired precision and performance.

3.6. Interactions

To clearly see how the search and diagnostic mobile agents duo works together, let's look at a brief example. In Figure 11, we see an extended version of Figure 9.

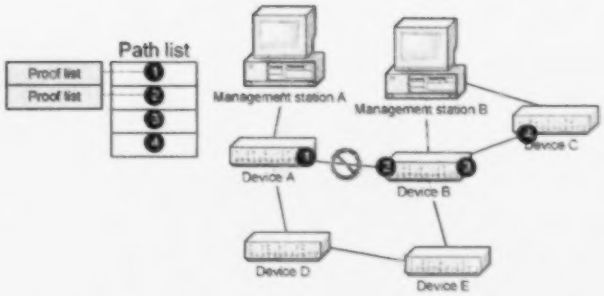

Figure 11. Simple example using search and diagnostic agents

The extension may be used by the search mobile agent to find an alternate path. In this case, the link between devices A and B is broken. Normally, this will cause one network interface on each of these devices to be automatically deactivated (the operational down state). If the diagnostic mobile agent is used alone, it will see only one deactivated interface. In a case where single failure is more likely, knowing that the other side is also automatically deactivated will help conclude that something between these two interfaces has gone wrong. If that other interface has been deactivated manually, it becomes apparent that only this interface is the problem. In our case, being able to find an alternate path enables the diagnostic agent to collect more facts about the failure and gives a better diagnostic. This path finding is handled by the search agent. In Figure 11, the diagnostic agent is stopped at device A. It launches a search agent that finds an alternative route using devices D and E. The search agent comes back to inform the mobile agent of this alternate path. The diagnostic agent may then use this alternate path to pursue its analysis phase on the other end of the failure. It is important to note that on this alternate path, the diagnostic agent does not collect information about devices D and E.

3.7. Fine Tuning

Our framework leaves mobile agent code on each management station. The mobile agent code is implemented in a class. The real mobile agents that move on the network inherit this

class without implementing any new functions. This way, we can tell *GrassHopper* to only move that lighter inheriting class and install the real code on each management station as core classes. GrassHopper never moves core elements such as its own platform classes and java native classes. What is then moved is only data and execution state, and this increases the responsiveness of the system and limits traffic. One drawback is that mobile agent code cannot be updated dynamically. This technique was inspired by the JAMES [20] architecture which uses a more complex system. It uses version checking and only downloads mobile agent's code as needed. This method is dynamic.

Another important technique that we used was to make sure to drop useless data before each movement. This practice is strongly suggested. By useless data, we mean information that is not used anymore, redundant or easy to get at a later time.

4. Experimental Results

In this section, we will first present in detail all four tests that were done and what goals we tried to achieve. Then, we will show the results of these tests. Finally, we will present our complete analysis of these results.

4.1. Tests

In our preliminary tests, it became apparent that using the diagnostic agent in conjunction with the search agent was an improvement in diagnostic precision, but had two serious side-effects: high response time and high total traffic on the network. We then chose to use two test networks to run our tests. The first one was used to show how easily the combination of the search and the diagnostic agent could locate and diagnose network fault. The easiness was based on the ability of mobile agents to enhance diagnostic precision over stationary mobile agents even if it comes

at a high price. A stationary agent, for this section, is a variant of a mobile agent that cannot move, and should not be taken for the stationary agent described earlier that is part of our framework. This first test network, shown in Figure 12, offers alternate paths.

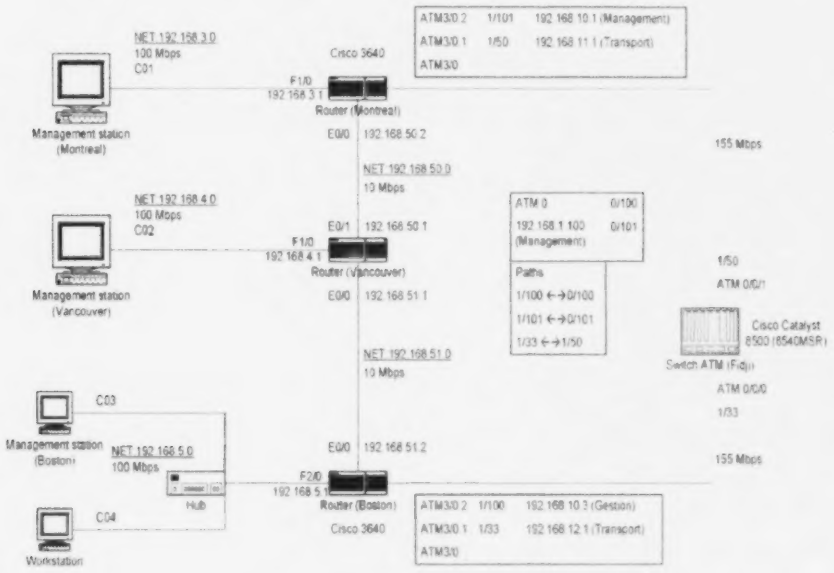

Figure 12. First test setup

This test network also has a second goal: evaluate qualitatively the advantages of using management mobile agents in real networks. It unravels areas where mobile agents are better suited than remote management: path finding and searching. This evaluation will be part of our analysis.

On this test network, we made a first test (Test #1) involving twenty random single faults. These twenty faults were simulated for the three following cases: diagnostic and search mobile agents (case #1), diagnostic mobile agent alone (case #2) and stationary diagnostic agent alone (case #3). In each case, we evaluated the precision of the diagnostic and we measured the response times. We also ran a test (Test #2) to demonstrate that the diagnostic agent (mobile or not) was able to know that the management system was in failure. This test is important to be

able to discriminate between a network failure involving a loss of service and a management system failure. Another simple test (Test #3) was run to ensure that the diagnostic agent does not mistakenly report network failure in a fully operational network.

The second test network's (Figure 13) goal was to evaluate mobile agents in terms of raw performance. While this test has already been done for various applications including network management, it was important to know how our framework rates against an equivalent remote approach. To achieve this, we built a test setup that enabled the mobile agent to be the closest possible to the device to manage. It is obvious that being closer to devices should lower the total traffic on the network while load balancing the charge on many devices. What is less obvious is calculating the penalty of moving network management code from one place to another. The second test network uses a restrained version of the first test network. The result of this subset is a network with only one route from one host to another. The mobile agent can manage each device from the closest management station. This test network makes it easier to test the performance of mobile agents against stationary agents. The last test (Test #4) was limited to failures that imply at least one migration for the mobile diagnostic agent. It does not use the search agent to provide a fair comparison. The mobile agent returns to the source to show its diagnostic, even if it has the ability to do its diagnostic at the destination. The stationary agent uses the same diagnostic algorithm, but is limited to no mobility at all.

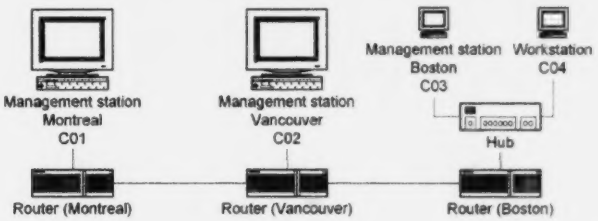

Figure 13. Second test setup

All tests were made with static routing and single failure scenario. These choices were made to lower the complexity of algorithms and mobile agents. Therefore, mobile agents presented in this paper are built for this type of scenario only. It is a limitation for our test, but it still shows possibilities of mobile agents for more typical scenarios. By single failure scenario, we mean that the tests are conducted with only one failure, but this failure may cause more than one alarm and more than one consequence on the network.

The content of the management table described in previous chapters is shown in Table 3.

Table 3. Management table used for all tests

Device	Management station
Router Montreal	Montreal
Router Vancouver	Vancouver
Router Boston	Boston
Switch Fidji	Montreal
Management station Montréal	Montreal
Management station Vancouver	Vancouver
Management station Boston	Boston

Specifications of the main components of test networks are shown in Table 4.

Table 4. Specification of main component of test networks

Description	Manageable	Operating system	Interfaces	Mobile agent platform
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbits	GrassHopper 2.2.4b JRE 1.3.1
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbits	GrassHopper 2.2.4b JRE 1.3.1
Intel Pentium 4 1.7Ghz 256meg	yes	Windows 2000 SP3	1x100 Mbits	GrassHopper 2.2.4b JRE 1.3.1
Router Cisco 3640	yes	IOS 12.2	1x100 Mbits 1x10 Mbits 1xATM155Mbits	not applicable
Router Cisco 3640	yes	IOS 12.1	1x100 Mbits 2x10 Mbits	not applicable
Router Cisco 3640	yes	IOS 12.2	1x100 Mbits 1x10 Mbits 1xATM155Mbits	not applicable
Switch Cisco Catalyst 8500	yes	IOS 12.0	2xATM 155Mbits	not applicable
Hub 4 ports	no	not applicable	4x100 Mbits	not applicable

By reviewing the components in Table 4, it appears clearly that the focus was on using different types of devices and different transport and management technologies. Our test network can be classified as a *heterogeneous network*.

4.2. Results

In the previous section, we exposed the two networks used for our tests as well as the details about four experimentations. In this section, we will show various results from these experimentations. The results of the first test are shown in Table 5. The results are for the three cases already stated which are: diagnostic and search mobile agents (case #1), diagnostic mobile agent alone (case #2) and stationary diagnostic agent alone (case #3).

Table 5. Results of test #1 for all three cases

Session	Management station		Network failure	Failure cause			Solution	Response time (s)		
	Source	Destination		unique	identified	near	rating	1	2	3
01	Mtl	Bos	Lien ATM0/0/1	**1**	1,2,3		**best**	70,87	**2,26**	3,26
02	Mtl	Bos	Lien ATM0/0/0	**1**	1,2,3		**best**	66,74	63,45	**41,17**
03	Mtl	Bos	Interface ATM0/0/1 on Fidji (admin down)	**1**	1,2,3		**best**	68,89	3,13	**2,14**
04	Mtl	Bos	Interface ATM0/0/0 on Fidji (admin down)	1,2,3	1,2,3		same	69,06	64,09	**40,62**
05	Mtl	Bos	Interface ATM3/0.1 on Bos (admin down)	**1**	**1**	2,3	**best**	60,80	63,48	**40,46**
06	Mtl	Bos	Link Bos station to Hub		1,2,3		same	85,87	37,97	**18,84**
07	Mtl	Bos	Interface ATM3/0.1 on Mtl (admin down)	1,2,3	1,2,3		same	67,97	3,61	**2,36**
08	Mtl	Van	Link Van router to Mtl	1	1,2,3		**best**	26,94	3,45	**2,38**
09	Mtl	Van	Interface E0/0 on Mtl (admin down)	1,2,3	1,2,3		same	50,30	2,41	**2,09**
10	Mtl	Van	Interface E0/1 on Van (admin down)	**1**	**1**	2,3	**best**	67,32	61,69	**39,11**
11	Mtl	Van	Router Van crashed		1,2,3		**best***	14,73	**2,28**	3,16
12	Mtl	Van	Service to reach not started on Van station	1,2,3	1,2,3		same	9,37	7,52	**6,96**
13	Mtl	Van	Van station crashed		1,2,3		same	141,18	39,23	**14,58**
14	Mtl	Van	Link Van station to Van router		1,2,3		same	69,45	29,80	**7,22**
15	Van	Mtl	Interface E0/0 (admin down)	**1**	1,2,3		**best**	70,03	62,30	**39,94**
16	Van	Mtl	Crash		**1**		**best**	51,45	67,21	**40,65**
17	Van	Bos	Interface E0/0 on Van (admin down)	1,2,3	1,2,3		same	55,62	5,34	**2,06**
18	Van	Bos	Interface F2/0 on Bos (admin down)	1,2,3	1,2,3		same	96,29	30,07	**6,02**
19	Bos	Van	Interface E0/0 on Bos (admin down)	1,2,3	1,2,3		same	57,65	2,66	**2,44**
20	Bos	Mtl	Link Mtl station to Mtl router		1,2,3		same	87,56	28,88	**6,09**

Varying results are identified by their case numbers. Each table in this section shows the best results for each session in bold. The solution rating column is based on a comparison of case #1 against the two other cases. It is interesting to note that case #2 and case #3 gave the same precision but not the same response times. Session #11 gave the same precision for each case, but we still rate the solution of case #11 as *best* because it returned more relevant information

about the problem than other cases. The *failure cause found* is separated in three columns. The first one, denoted *unique*, indicates that the exact cause of the failure was found and presented as the unique cause. This is the ideal diagnostic for a network administrator. The second one, noted *identified*, indicates that the cause was identified but lies among a series of other relevant but not exact causes. It may also indicate that the cause was not found precisely, but the right device was found. The last one, called *near*, indicates that the diagnostic was wrong, but near the cause of the failure. By using the search and diagnostic mobile agents, we always got a better or equivalent precision against the diagnostic agent alone, mobile or not. However, by using the search mobile agent, we significantly increased the response time and the total traffic on the network. The total traffic was not measured for each session but tends to be, on average, eight times greater when the search agent is used.

The next two tests are shown in Table 6. As we can see, the diagnostic mobile agent behaved like expected. The diagnostic agent is able to see a difference between a network failure and a management system failure. Also, it behaved as expected in sessions without any network failure.

Table 6. Special case experiments (Test #2 and Test #3)

Session	Management Station		Network failure	Response time (s)	Cause found
	Source	Destination			
01	Mtl	Bos	None	12,09	None
02	Mtl	Van	None	7,86	None
03	Van	Mtl	None	12,40	None
04	Van	Bos	None	6,69	None
05	Bos	Mtl	None	10,02	None
06	Bos	Van	None	8,34	None
07	Mtl	Bos	Fidji management system disabled	189,24	Yes
08	Mtl	Van	Mtl management system disabled	194,02	Yes
09	Van	Bos	Bos management system disabled	195,08	Yes
10	Bos	Mtl	Bos management system disabled	195,56	Yes
11	Bos	Van	Van management system disabled	203,43	Yes

All results we have shown until now on were to evaluate mobile agent technology advantages over remote solutions and stationary agents. The next results give a better idea of the load imposed on networks by mobile agents against stationary agents.

The traffic measurements are shown in Table 7 for the diagnostic mobile agent and in Table 8 for the diagnostic stationary agent. Both agents provide the same identification precision.

Table 7. Traffic measurements for diagnostic mobile agent (Test #4)

Session	Traffic (bytes)				Response time (s)
	Mtl	Van	Bos	Total	
01	**46092**	74988	12062	133142	47,18
02	43714	49876	0	93590	10,13
03	**43206**	59861	42380	145447	16,89
04	**52936**	113634	27510	194080	32,90
05	46541	46540	1342	94423	65,39
06	**41099**	59339	41469	141907	11,24

Table 8. Traffic measurements for diagnostic stationary agent (Test #4)

Session	Traffic (bytes)				Response time (s)
	Mtl	Van	Bos	Total	
01	62209	**29051**	**11642**	**102902**	**15,62**
02	**41895**	**12313**	0	**54208**	**4,06**
03	70328	**35707**	**22589**	**128624**	**9,72**
04	104449	**59042**	**27120**	**190611**	**7,39**
05	**35584**	**10326**	**703**	**46613**	**39,03**
06	70775	**36408**	**23988**	**131171**	**6,65**

For clarity, in tables 7 and 8, values are bold for the best values between both tables. The stationary agent always got a better response time. It also generates less total traffic in each case and less traffic around almost all routers.

Table 9 shows total traffic value for mobile and stationary agents for each session and Table 10 compares each traffic value around each router. To measure traffic values around router, we used *out bytes* value on each interface of a given router.

Table 9. Total traffic for each session for test #4

Session	Total traffic (bytes)		Traffic variation
	Mobile	Stationary	
01	133142	102902	29,4%
02	93590	54208	72,6%
03	145447	128624	13,1%
04	194080	190611	1,8%
05	94423	46613	102,6%
06	141907	131171	8,2%

Table 10. Average traffic for each router and overall for test #4

	Traffic (bytes)			
	Mtl	Van	Bos	Total
Mobile agent	45598	67373	20794	133765
Stationary agent	64207	30475	14340	109022
Variation	40,8%	-54,8%	-31,0%	-18,5%

Table 9 still shows a performance advantage for the stationary agent, but with one session being almost equal (session 04 with less than 2% of variation) and two sessions with small variation (sessions 03 and 06). Table 10 shows interesting statistics when we look at average values. The stationary agent only produces 18.5% less traffic on average for all sessions. Also, on router *Montreal*, the diagnostic mobile agent has a clear advantage over the stationary agent that is not seen in other routers.

4.3. Analysis and Discussion

One drawback of our work is the relatively small scale of the experiment. However, our results show promising performance and usability for mobile agents in network management applications. It appears that a large-scale network will benefit even more from mobile agents than our small-scale experiment.

At first glance, the stationary agent seems to be a better solution if we only consider performance. But, we still think that mobile agents will be a better solution even in this area of comparison considering the small test network used. We base our statement on the following

findings: the total traffic value for hardest network tasks is almost the same for diagnostic stationary and mobile agents (Table 9) and the average network load on the first router is high for the stationary agent (Table 10). By hardest network tasks, we mean tasks that involve lots of information querying. Task 03, 04 and 06 are good examples. In these cases, the diagnostic agent needs more information about the network and the cost of movement is attenuated by the heavier network management traffic. By contrast, session 05 of Test #4 (Table 9) involves few information querying. It is also the worst performance of the diagnostic mobile agent against the stationary agent because the cost of movement is high compared to the network management traffic involved. In a small network, the cost of remote management is not high enough to switch to mobile agent management. But our results let us think that mobile agents will unveil a better performance on bigger networks with most demanding and many simultaneous network tasks [1,14]. Another point of interest is the network load imposed on equipment near the remote management station. Results of Table 10 show that if we would like to execute more than one management task at the same time, the first router may become a bottleneck. It is now a fact, that mobile agents have a higher response time and may create more traffic overall. But, one main interest about them is their potential of repartition of the network load on the whole network [10,12]. One bad point about our results is the high traffic variation of 54.8% imposed by mobile agents on the Vancouver router. This is mainly due to the diagnostic agent having to return home and may be optimized further by only sending a report to the source if the result is needed at the source.

From here, we did a performance analysis. We are also interested, in the mobile agent domain of research, to find tasks that are enabled only by mobile agents. So far, it seems that there is no such task. However, what we saw in our experiments are tasks that are more easily executed by

mobile agents. For example, we have demonstrated that the search and diagnostic mobile agents were able to find more precisely a cause of a network failure by finding alternate paths to gather more data about the failure (Table 5). In seven of the twenty random failures, they were able to find a better solution. In these seven cases, mobile agents were able to gain access easily to interesting information about the failure that remote management was not able to see. This precision seems to be related to the number of alternate paths a network offers. Also, another point of interest is that diagnostic mobile agents can still work efficiently in an unreliable network whereas its remote counterpart may have a hard time doing the same task. Mobile agents are more suited for automation. Finally, they offer better ways to use existing network management utilities and facilities by their ability to access local resources directly and efficiently.

5. Conclusion

In this paper, we presented, after a brief introduction, our network management framework using mobile agents and we presented important parts of that framework. We then described two mobile agents that we used to validate and prove network management capabilities of mobile agents in a real heterogeneous network. These mobile agents do not provide the best way to make fault management, but our goal was to show the interest of using mobile agents for network management. We then presented our experimental results.

Our framework has many limitations that must be overtaken before using mobile agents for real applications. Security and fault-tolerance are probably the greatest issues that mobile agent systems must face. Security is important when it comes to large-scale networks where services must be maintained and data on the network must be protected. The framework is also still limited in its management capabilities. It only offers a small set of management functions that

should be improved before it could be ready to be used by mobile agents for any network task. The projected performance improvement is not yet proven for mobile agents of our framework. The diagnostic mobile agent is also limited in the kind of network and errors it can handle.

Limitations stated in the last paragraph can all constitute future ways to improve the current framework. In [3], Borselius offers an overview of security issues and measures for mobile agent systems. We also already stated some possible optimization and interesting papers in this article that could be used to further improve this framework. The set of management functionalities that it can handle could be enhanced. By enhancing this set, it would become possible to build mobile agents that execute network management tasks other than network fault localization and diagnostic. Mobile agents presented in this paper could still be improved in many ways. For example, the diagnostic could handle much more failure causes and could be adapted for dynamic routing protocols. The diagnostic algorithm could learn from past errors instead of being tied to a deterministic expert system. Even the expert system can be improved using better expert system techniques, scripts and rules. Network fault location and diagnostic are two tasks that network administrators would want to be automated efficiently and confidently. Since mobile agents we presented are limited, there is still a lot of work to do only in that area.

The topic of determining optimal associations between components and management stations is a research field by itself. Management tables could be updated when network topology and network parameters change. This update could be done based on performance analysis that could also be managed by mobile agents. Even the management station positioning could be improved by enabling stations to move, being themselves mobile agents [10].

Mobile agents, even if they do not enable any new possibilities, seem to at least offer a new and interesting way of doing network management. They seem to be the best way to handle the complexity of today's network. Based on our research, we are convinced that network management will benefit from mobile agent technology.

References

[1] M. Baldi, S. Gai and G.P. Picco, "Exploiting Code Mobility in Decentralized and Flexible Network Management," *Proc. Mobile Agents. First International Workshop*, Berlin, Germany, pp. 13-26, 1997.

[2] B. Baràn and R. Sosa, "A New Approach for AntNet Routing," *Proc. Ninth International Conference on Computer Communications and Networks*, Las Vegas, NV, USA, pp. 303-308, 2000.

[3] N. Borselius, "Mobile Agent Security," *Electronics & Communication Engineering Journal*, vol. 14, no.5, pp. 211-218, 2002.

[4] J. Boyer, B. Pagurek and T. White, "Methodologies for PVC Configuration in Heterogeneous ATM Environments Using Intelligent Mobile Agents," *Proc. of MATA'99, Mobile Agents for Telecommunications Applications*, Ottawa, Canada, 1999.

[5] Carleton University, The Network Management and AI Laboratory. http://www.sce.carleton.ca.

[6] D. Dasgupta, H. Brian, "Mobile Security Agents for Network Traffic Analysis," *Proc. DARPA Information Survivability Conference and Exposition II*, vol. 2, 2001, Anaheim, CA, USA, pp. 332-340, 2001.

[7] G. Di Caro, M. Dorigo , "Ant Colonies for Adaptive Routing in Packet-Switched Communications Networks," *Proc. Parallel Problem Solving from Nature - PPSN V*, Amsterdam, Netherlands, pp. 673-682, 1998.

[8] Timon C. Du, Eldon Y. Li and An-Pin Chang, "Mobile Agents in Distributed Network Management," *Communications of the ACM*, vol. 46, no.7, pp. 127-137, 2003.

[9] D. Gavalas, D. Greenwood, M. Ghanbari and M.J. O'Mahony, "An Infrastructure for Distributed and Dynamic Network Management Based on Mobile Agent Technology," *Proc. IEEE International Conference on Communications*, Vancouver, Canada, vol. 2, pp. 1362-1366, 1999.

[10] D. Gavalas, D. Greenwood, M. Ghanbari and M. O'Mahony, "Hierarchical Network Management: A Scalable and Dynamic Mobile Agent-Based Approach," *Computer Networks*, vol. 38, no. 6, pp. 693-711, 2002.

[11] G. Helmer, J.S.K. Wong, V. Honavar and L. Miller, "Automated Discovery of Concise Predictive Rules for Intrusion Detection," *Journal of Systems and Software*, vol. 60, no. 3, pp. 165-75, 2002.

[12] A. Liotta, G. Pavlou and G. Knight, "Exploiting Agent Mobility for Large-Scale Network Monitoring," *IEEE Network*, vol. 16, no. 3, pp. 7-15, 2002.

[13] B. Pagurek, Y. Wang and T. White, "Integration of Mobile Agents with SNMP: Why and How," *Proc. IEEE/IFIP Network Operations and Management Symposium*, pp. 609-622, 2000.

[14] A. Puliafito and O. Tomarchio, "Advanced Network Management Functionalities Through the Use of Mobile Software Agents," *Proc. Intelligent Agents for*

Telecommunication Applications. Third International Workshop, IATA'99, Berlin, Germany: Springer-Verlag, pp. 33-45, 1999.

[15] D. Putzolu, S. Bakshi, S. Yadav and R. Yavatkar, "The Phoenix Framework: A Practical Architecture for Programmable Networks," *IEEE Communications*, vol. 38, no. 3, pp. 160-5, 2000.

[16] D. Rossier and R. Scheurer, "An Ecosystem-inspired Mobile Agent Middleware for Active Network Management," *Proc. 4th International Workshop on Mobile Agents for Telecommunication Applications*, pp. 73-82, 2002.

[17] M.G. Rubinstein, O. Duarte and G. Pujolle, "Scalability of a Network Management Application Based on Mobile Agents," Technical Report, [online] http://www.gta.ufrj.br/~rubi/public.html, 2002.

[18] M.G. Rubinstein, O. Duarte, G. Pujolle, "Reducing the Response Time in Network Management by Using Multiple Mobile Agents," *Proc. Third International Conference on Management of Multimedia Networks and Services*, Norwell, MA, USA, Kluwer Academic Publishers, pp. 253-265, 2000.

[19] R. Schoonderwoerd, O. Holland and J. Bruten, "Ant-Like Agent for Load Balancing in Telecommunication Networks," Technical Report, Hewlett-Packard Laboratories, Bristol-England, 1997.

[20] L.M. Silva, P. Simoes, G. Soares, P. Martins, V. Batista, C. Renato, L. Almeida and N. Stohr, "JAMES: A Platform of Mobile Agents for the Management of Telecommunication Networks," *Proc. Third International Workshop, IATA'99*, Springer-Verlag, pp. 77-95, 1999.

[21] T. White and B. Pagurek, "Towards Multi-Swarm Problem Solving in Networks," *Proc. Third International Conference on Multi-Agent Systems ICMAS '98*, pp 333-340, July 1998.

[22] T. White and B. Pagurek, "Distributed Fault Location in Networks using Learning Mobile Agents," *Proc. of PRIMA '99*, Springer-Verlag, pp. 182-196, 1999.

[23] T. White, B. Pagurek, and A. Bieszczad, "Network Modeling For Management Applications Using Intelligent Mobile Agents." *Journal of Network and Systems Management*, vol. 7, no.3, pp. 295-321, 1999.

[24] M. Zapf, K. Herrmann and K. Geihs, "Decentralized SNMP management with mobile agents," *Proc. of the Sixth IFIP/IEEE International Symposium on Distributed Management for the Networked Millenium 1999*, pp. 623-635, 1999.